Section 2: ASP.NET MVC fundamentals

ActionResult : This is the base class for all action results in asp.net mvc

View is the helper method derived from base Controller class.

Its

Vidly : Its a video rental store application ,

It has two roles , Admin, staff member

Can Login using facebook.

Extends the application and add support for google nd facebook

it includes business rules,

MVC : Architectural patterns for implementing interfaces.

Model: Model represents application data and behaviour in terms of its problem domain and independant of UI.

The model classes consists of properties and methods that purely represents the application state and rules.

We can take these model classes and can use them in a different application like desktop or mobile app.

since Domain classes are Plain Old CLR Objects or POCOS.

It doesnot have any dependancy on external framework.

View :HTML markup displayed to the user.

Controller : Is responsible for handling http requests.

Router : when a request comes in the application a controller will be selected for handling that request.

Selecting the right controller is the responsibility of the router.

Git : source Control

App_Data : Where tha datadase file is stored.

App_start : which contains classes which will called when tha application started.

RouteConfig : this is the configuration of our routing rules.

The default route name is default

url : Contains the url pattern

If the url matches this pattern.

The default route contains some dafult values.

So if the url doesnt have values for any of this pattern , like controller, action , or id this it will take the deafult values.

If the url have only controller nd no action then it will take the default value from the default url and complete the request.

Content : We store the css files, images and any other client side assets.

Controller :

Account : Wihch contains actions for signup, login, logout etc*

Home : Which represents the home page

Manage : It provides a number of actions for handling the requests around users profile. Like changing password, enabling two factor authentication, using social logins and so on.

Models : all the domain classes will be here.

Scripts: we store the javascipt files.

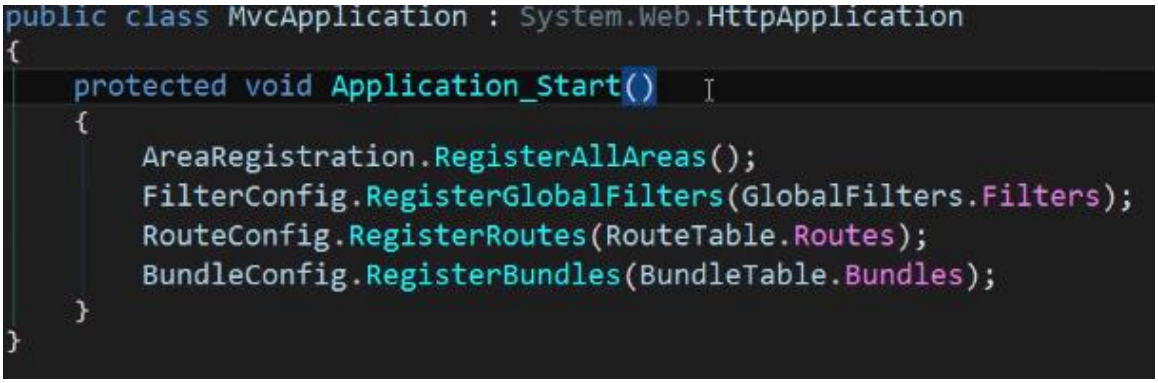
Views :

Shared:

It includes a folder called shared, which contains views that can be shared across different controllers

Favicon : is the icon of the application displayed in the browser.

Global.asax :



```
public class MvcApplication : System.Web.HttpApplication
{
    protected void Application_Start()
    {
        AreaRegistration.RegisterAllAreas();
        FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
        RouteConfig.RegisterRoutes(RouteTable.Routes);
        BundleConfig.RegisterBundles(BundleTable.Bundles);
    }
}
```

When the application is started this method is called

AppIcation_Start : It registers few things. eg : Routes

When the application is started we tell the runtime these are the routes for the application

Package.config: Which is used by nuget pkg manager

Nuget : Its a package manager similar to node package manager

Package managers are used to manage dependencies of our application.

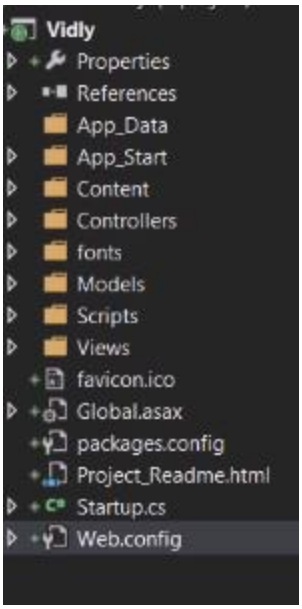
If our application has dependencies on five different applications, instead of going to five different websites and downloading these libraries we use package manager to download these dependencies from the central repository.

In future if one of these libraries has a newer version again we can use the package managerto upgrade one or more of the existing packages

Startup.cs: Which is a new approach microsoft has taking.

In the .net core MS has dropped the global.asax files and applied all those logics into the startup.cs

Web.config : Which is an xml which includes the configuration for application.



Code snippet : prop

Template : which used for autogenerating code which we called scffolding.

In the view we can use directive to get data as :

```
@model Vidly_.Models.Movie;
```

To use this data we can use @Model

Bootstrap :

To use a different bootstrap style we can use bootswatch.com

lumen

css(not min)

Bundle.css : It compresses and combine the javascript files , it reduces the number of http requests required to get the assests when the page is loaded

Action Result:

Its the base class for all return type of Action method

View is the helper method of action method derived from controller class

Type	Helper Method
ViewResult	View()
PartialViewResult	PartialView()
ContentResult	Content()
RedirectResult	Redirect()
RedirectToRouteResult	RedirectToAction()
JsonResult	Json()
FileResult	File()
HttpNotFoundResult	HttpNotFound()
EmptyResult	

Action Parameters :

MVC maps request data into paramater values for action methods.

These parameter values can be embeded in th url

or it can be a query string or the form data.

```
public ActionResult Edit(int id)
{
    return Content("Id :" + id);
}
```

<http://localhost:63913/Movies/Edit/1 //> This works since the default parameter name in the route config is id, If we change the paramater name then we need to use query string.

http://localhost:63913/Movies/Edit?id=39

To make parameter optional :

To make parameter nullable we need tomake the parameter optional.

```
public ActionResult Index(int? pageIndex, string sortBy)
{
    if(!pageIndex.HasValue)
    {
        pageIndex = 1;
    }
    if(string.IsNullOrWhiteSpace(sortBy))
    {
        sortBy = "Name";
    }
    return Content(string.Format("PageIndex ={0} , sortBy={1}", pageIndex, sortBy));
}
```

Here since int is a non nullable value we can use "?" to make it nullable.

Code Snippet for creating new Action:

MVCAction4 :Tab

Custom Routing : (old method)

In route.config

```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
    routes.MapRoute(
         "MoviesByReleaseDate",
         "Movies/ByDate/{releaseDate}",
         new { controller = "Movies", action = "ByReleaseDate"}
    );
```

MoviesByReleaseDate is the route name

Movies/ByDate/{releaseDate} - is the url structure

{releaseDate} - is the parameter value

Adding constraints for paramaters:

```
routes.MapRoute(
    "MoviesByReleaseDate",
    "movies/released/{year}/{month}",
    new { controller = "Movies", action = "ByReleaseDate" },
    new { year = @"\d{4}", month = @"\d{2}" });
```

Attribute Routing :

Drawbacks of custom routing :

We need to go back and forth between action and routeconfig.

If more custom routing that will be a mess in the route config.

if we change the name of the action it will not update in the route config, so the code become fragile.

Inorder to implement attribute routing we need to add

```
routes.MapMvcAttributeRoutes();
```

in th route config file.

```
[Route("Movies/ByDate/{releaseDate}")]
0 references | 0 changes | 0 authors, 0 changes
public ActionResult ByReleaseDate(string releaseDate)
{
    return Content("Relase Date :"+releaseDate);
}
```

Adding constraints to attribute routing:

```
[Route("Movies/ByDate/{year}/{month:regex(\\d{2}):range(1,12)}")]
0 references | 0 changes | 0 authors, 0 changes
public ActionResult ByReleaseDate(int year, int month)
{
    return Content(string.Format("Relase Date {0} / {1}:",month,year));
}
```

Google : asp.net mvc attribute route constraints

Passing data to views:

ViewData dictionary:

```
public ActionResult Random()
{
    var movie = new Movie() { Name = "Shrek" };
    ViewData["Movie"] = movie;
    return View();
}
```

View Code :

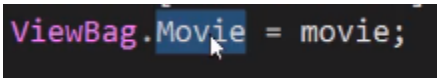
```
@using Vidly_.Models;
@model Vidly_.Models.Movie

@{
    ViewBag.Title = "Random";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>@(((Movie)ViewData["Movie"]).Name)</h2>
```

The drawbacks of ViewData:

We need casting , since the each item in the dictionary is of type object, so we need casting.

More over if we change the magic string "Movie" used in the controller change it will not change in the view, we will get a null reference exception.

ViewBag : which is a dynamic type,.



ViewBag comes with a magic property, which will add the Movie class at runtime, which doesnt have a compile time safety.

It also require type casting

```
public ActionResult Random()
{
    var movie = new Movie() { Name = "Shrek" };
    ViewBag.Movie = movie;
    return View();
}
```

View:

```
@using Vidly_.Models;
@model Vidly_.Models.Movie

@{
    ViewBag.Title = "Random";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>@(((Movie)(ViewBag.Movie)).Name) </h2>
```

So its better to use View(model)

```
public ActionResult Random()
{
    var movie = new Movie() { Name = "Shrek" };
    return View(movie);
}
```

ViewModel:

```
public ActionResult Random()
{
    var movie = new Movie() { Name = "Shrek"
    var customers = new List<Customer>()
    { new Models.Customer()
    {Name="Sheeba" },
    new Models.Customer()
    {Name="Shibu" }
    };
    var viewModel = new RandomMovieViewModel
    {
        Movie = movie,
        Customer = customers
    };
    return View(viewModel);
}
```

ViewModel Class

```
public class RandomMovieViewModel
{
    1 reference | 0 changes | 0 authors, 0 changes
    public Movie Movie { get; set; }
    1 reference | 0 changes | 0 authors, 0 changes
    public List<Customer> Customer { get; set; }
}
```

View :

```
@using Vidly_.ViewModel;
@model Vidly_.ViewModel.RandomMovieViewModel

@{
    ViewBag.Title = "Random Movie";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>@Model.Movie.Name</h2>
```

Razor View Syntax:

```
@using Vidly_.ViewModel;
@model Vidly_.ViewModel.RandomMovieViewModel

@{
    ViewBag.Title = "Random Movie";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>@Model.Movie.Name</h2>
@if (Model.Customer.Count == 0)
{
    <text>No one has rented this movie</text>
}
else
{
<ul>
    @foreach (var customer in Model.Customer)
    {
        <li>@customer.Name</li>
    }
</ul>
}
```

<text> : is not standard html, but Razor can understand this.

Adding style on Condition:

```
@{
    var className = Model.Customer.Count > 5 ? "popular" : null;
}

<h2 class="@className">@Model.Movie.Name</h2>
```

Here when the customer count greater than 5 then header will get style "popular"

Comments : @* this is a comment *@

Partial View:

View ->Shared ->Layout.cs

Partial View is a small view that can be used in multiple page or we can use partial view to split a large view into smaller views.

@RenderBody() : in Layout.cs : What we put in view will render here

To create partial view : shared -> add view-> (add _ before view name for partial view)

To render the partial View :

@Html :Its the property of the view, which is of type Html helper , which is a class which contains a number of methods to work with html.

```
@Html.Partial("NavBar");
```

This Partial method expects a model

If we are not providing the model as a parameter, the Partial will take the view model as the parameter.

If we pass a model as a parameter, then the partail view will use that model as a parameter.

https://github.com/mosh-hamedani/vidly-mvc-5/commits/master

Entity Framework :

Its a tool we used to access the database.

Its an ORM

It mapps application data into database.

With ORM we dont need to open connection , do mapping, close connection

EFprovides a class named DBContext

Its a gateway into the database.

DbSet :Represents tables in our database.

We use LINQ to query the dbsets

EF translates LINQ queries to sql queries at runtime.

WOrkflows of EF :

Database first, Code first

Codefirst is more productive

We dont have to waste our time wuth table designers.

We need to create manually the change script inorder to deploy the database.

We will get full versioning of database, we can migrate to any version at anytime.

We can swtch back to the older version, we use migrations for that.

Its easy to create integration test database.

Code first Migrations:

For any changes required for the database, we create migrations and run it in the database.

For the first time we need to enable migrations

Note : Need to add ADO.NET Entity Data Model

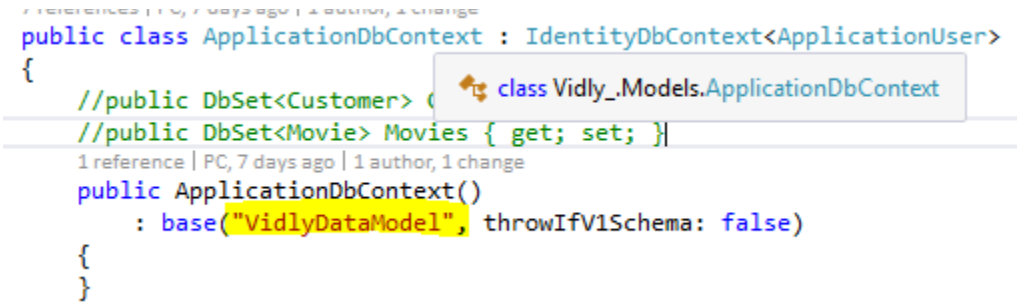
create connection string using this datamodel

pm> enable-migrations (not case sensitive)

After executing this we can see a new folder named migrations.

Then create migration by using the command

Note : check for the ADO.NET datamodel name here and in connection string.

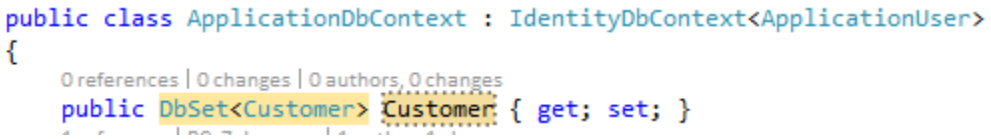


pm> Add-migration InitialModel

Here need to check whether the Up() contains CreateTable("dbo.aspnetroles")

These tables are coming from ASP.net identity which is used for authentication and authorisation.

Inorder to the tables into DBContext, we need to add property into IdentityModel.cs



To generate database

pm> update-database

To add field: Using Migrations

Add property into the class

```
public class Customer
{
    3 references | 0 changes | 0 authors, 0 changes
    public int Id { get; set; }
    4 references | 0 changes | 0 authors, 0 changes
    public string Name { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public bool IsSubscribedToNewsletter { get; set; }
}
```

```
PM> add-migration AddIsSubscribedToCustomer
```

```
public partial class AddIsSubscribedToCustomer : DbMigration
{
    1 reference | 0 changes | 0 authors, 0 changes
    public override void Up()
    {
        AddColumn("dbo.Customers", "IsSubscribedToNewsletter", c => c.Boolean(nullable: fa
    }
```

Add migrations : Implement small changes and add migrations.

Navigation Type :

```
9 references | 0 changes | 0 authors, 0 changes
public class Customer
{
    3 references | 0 changes | 0 authors, 0 changes
    public int Id { get; set; }
    4 references | 0 changes | 0 authors, 0 changes
    public string Name { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public bool IsSubscribedToNewsletter { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public MembershipType MembershipType { get; set; }
}
```

Navigation Property allows us to navigate from one type to another.

It will be useful when we want to load one object and its related object together.

Creating a foreign key in the class :

Sometimes we dont need the entire object , then we will create only one property of the related object, like foreign key.

Seeding the database:

We need to add Membershiptype data. This need to be consistent across different evironments, such as development, test, and production.

To achieve this consistency we need to use migration, since its a database first

workflow we cant touch the database.

Any changes to the database or database schema is properly recorded in the migration

```
public override void Up()
{
    Sql("Insert into membershiptypes(Id,SignUpFee,Durationinmonths,discountrate) values(1,0,0,0)");
    Sql("Insert into membershiptypes(Id,SignUpFee,Durationinmonths,discountrate) values(2,30,1,10)");
    Sql("Insert into membershiptypes(Id,SignUpFee,Durationinmonths,discountrate) values(3,90,3,15)");
    Sql("Insert into membershiptypes(Id,SignUpFee,Durationinmonths,discountrate) values(4,300,12,20)");
}
```

Overriding Conventions : (Data Annotations)

```
[Required]
[MaxLength(255)]
4 references | 0 changes | 0 authors, 0 changes
public string Name { get; set; }
0 references | 0 changes | 0 authors, 0 changes
```

Here string was nullable, since its a reference type and there were no limitation for the characters that can enter.

To override those conventions, we use data anotation attribute.

For this also we need to cerate migration

Querying Objects :

```
public class CustomerController : Controller
{
    private ApplicationDbContext _context;
    0 references | 0 changes | 0 authors, 0 changes
    public CustomerController()
    {
        _context = new ApplicationDbContext();
    }
    4 references | 0 changes | 0 authors, 0 changes
    protected override void Dispose(bool disposing)
    {
        _context.Dispose();
    }
```

Inorder to get data from database, we need to create ApplicationDbContext object.

Since its a disposible object we override Dispose() method of the base class.

ApplicationDbCOntext class is in Model -> IdentityModel class

Deffered execution:

Entity Framework is doing deferred execution.

```
var customers= _context.Customers;
```

ie when the above line is executed it will not immediately execute the query.

It will execute only when iterating through the customer object.

we can make this to execute the linq immediately by using .ToList() function.

Eager Loading:

```
<td>
    @customer.MembershipType
</td>
```

This code will provide a null reference exception, because it will provide only the current object, not the related object,

Inorder to get the related object we need to do eager loading.

ie We need to change the controller code from this to

```
public ViewResult Index()
{
    var customers = _context.Customers.ToList();
    return View(customers);
}
```

this

```
public ViewResult Index()
{
    var customers = _context.Customers.Include(c=>c.MembershipType).ToList();
    return View(customers);
}
```

We need to include System.Data.Entity;

Shortcut to PM console : Tools ->Options -> Environment -> Keyboard ->(search in) -> show commands containing -> set keys

The model backing the 'ApplicationDbContext' context has changed since the database was created. Consider using Code First Migrations to update the database

:

IdentityModels.cs ->In ApplicationDBContext constructor : Add ->

```
Database.SetInitializer<ApplicationDbContext>(null);
```

Add Column BirthDate to Customer :

Add property to Customer class

PM> Add-migration `AddColumnBirthDateToCustomer`

```
public partial class AddColumnBirthDateToCustomer : DbMigration
{
    8 references | 0 changes | 0 authors, 0 changes
    public override void Up()
    {
        AddColumn("dbo.Customers", "BirthDate", c => c.DateTime(nullable: true));
    }

    8 references | 0 changes | 0 authors, 0 changes
    public override void Down()
    {
        DropColumn("dbo.Customers", "BirthDate");
    }
}
```

```
Sql("insert into Movies values('HangOver','Comedy','12/03/1986',GETDATE(),5)");
```

Building Forms

```
@Html.BeginForm("Create","Customer")
```

This will render a begin <form> tag , not the </form> tag will included.

Inorder to automatically dispose this we use using

@using(Html.BeginForm("Create" , "Customer")

{

}

Markup used to render modern and responsive form is, that bootstrap understands

The input fields we can wrap in

(lable and input field)

    @Html.LabelFor(

```
<div class="form-group">
    @Html.LabelFor(m=>m.Name)
    @Html.TextBoxFor(m => m.Name)
</div>
```

If we use html controls mvc supports validation automatically

```
<input data-val="true" data-val-maxlength="The field Name must be a string
or array type with a maximum length of '255'." data-val-maxlength-max=
"255" data-val-required="The Name field is required." id="Name" name=
"Name" type="text" value> == $0
```

f12 screen : Here the validation has come since we have used the html helper methods, if we use raw html we had to implement again validation.

we can add an anonymous object into the html helper method as paramater.

this parameter will get render as html attribute.

```
@Html.TextBoxFor(m => m.Name,new {@class="form-control" })
```

```
https://getbootstrap.com/docs/3.3/css/#forms

for checkbox you need to add a different style, check in the link above.
```

Label

LabelFor for changing the Displayed letter:

We can apply data annotation to the property.

```
[Display(Name="Date of Birth")]
2 references | 0 changes | 0 authors, 0 changes
public DateTime? BirthDate { get; set; }
```

The drawback for this approach is every time we need to compile the code.

Alternative way is in the view

Manually add <label>Date of Birth </label> in the view

The drawback for this approach is when we click on the label in labelfor it will autofocus the textbox

If we use the html label it will not autofocus the textbox.

Inorder to overcome this we can add "for" attribute to the html label

```
<label for="BirthDate"> Date of Birth</label>
@Html.TextBoxFor(m => m.BirthDate, new { @class = "form-control" })
```

The problem for this approach is we need to remember to update the magic string while changing the property.

Dropdown list :

We need both Customer and MembershipType for that particular view

so we can use viewmodel

IList<MembershipType> or IEnumerable<MembershipType>

but IEnumerable<MembershipType> is more better

since every function is using IEnumerable in view

IList implements IEnumerable. So the code will be more loosely coupled.

View Model

```
public class NewCustomerViewModel
{
    1 reference | 0 changes | 0 authors, 0 changes
    public IEnumerable<MembershipType> MembershipTypes { get; set; }
    5 references | 0 changes | 0 authors, 0 changes
    public Customer Customer { get; set; }
}
```

```
public ActionResult New()
{
    var membershipTypes = _context.MembershipTypes.ToList();

    var viewModel = new NewCustomerViewModel()
    {
        MembershipTypes = membershipTypes,

    };
    return View(viewModel);
}
```

```
@model Vidly_.ViewModel.NewCustomerViewModel
@{
    ViewBag.Title = "New";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>New Customer</h2>
@using (Html.BeginForm("Create", "Customer"))
{
    <div class="form-group">
        @Html.LabelFor(m=>m.Customer.Name)
        @Html.TextBoxFor(m => m.Customer.Name, new {@class="form-control" })
    </div>
    <div class="form-group">
        @Html.LabelFor(m => m.Customer.BirthDate)
        @Html.TextBoxFor(m => m.Customer.BirthDate, new { @class = "form-control" })
    </div>
    <div class="checkbox">
        <label>
            @Html.CheckBoxFor(m=>m.Customer.IsSubscribedToNewsletter) Is Subscribed to newsletter?
        </label>
    </div>
    <div class="form-group">
        @Html.LabelFor(m => m.Customer.MembershipTypeId)
        @Html.DropDownListFor(m => m.Customer.MembershipTypeId,new SelectList (Model.MembershipTypes,"Id","Name" ),
        "", new { @class = "form-control" })
    </div>
    <button type="submit" class="btn btn-default">Submit</button>
}
```

Model Binding :

MVC automatically maps request data to the object.

Accessing request values using the Request object is a cumbersome and time wasting activity.

With model binding, MVC framework converts the http request values (from query string or form collection) to action method parameters. These parameters can be of primitive type or complex type.

Save Data:

```
[HttpPost]
0 references | 0 changes | 0 authors, 0 changes
public ActionResult Create(Customer customer)
{
    _context.Customers.Add(customer);
    _context.SaveChanges();
    return RedirectToAction("Index", "Customer");
}
```

Inorder to add data into DB we need to first add into DBContext

After this stment, it will be just in memory, will not get added into database.

DBContext as a change tracker mechanism, anytime you add an object to it or modify or remove an existing object ot will mark as modified or deleted

To persist these changes we need to call _context.SaveChanges();

It will generate sql statements at run time and will execute,

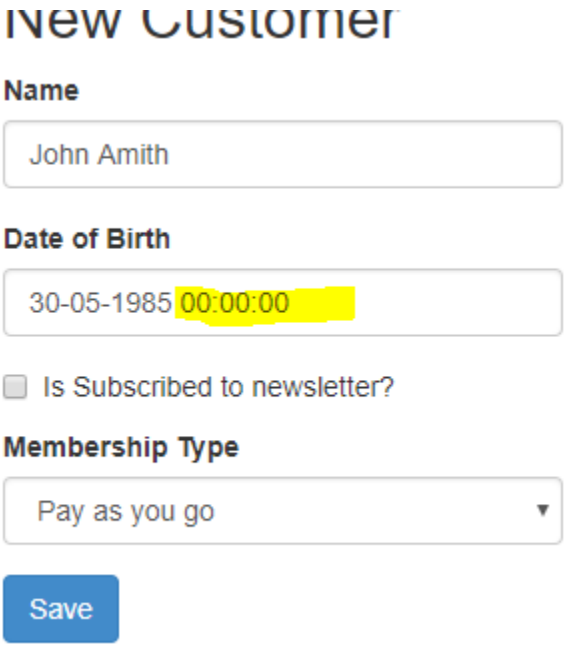
All these statements will binded inside a transaction.

All changes will execute together , or nothing will execute.

ActionLink :

```
@Html.ActionLink(customer.Name, "Edit","Customer",new { id=customer.Id},null)
```

```
customer.Name, -> Link text

Edit -> Action name

Customer -> Controller name

new { id=customer.Id},  -> (Route values) , values passing through action link as the Controller action Parameter

null -> Html Attribute
```



Here we dont want date in this format

So we can one of the TextBoxFor overload for this :

@Html.TextBoxFor(m => m.Customer.BirthDate,"{ 0:d MMM yyyy}", new { @class = "form-control" })

"{0: d MMM yyy}" -> this is format string

Updating Data

```
@using (Html.BeginForm("Create", "Customers"))
```

Its the target action. (create)

CustomerForm.cshtml

```
@model Vidly_.ViewModel.NewCustomerFormViewModel

@{
    ViewBag.Title = "New";
    Layout = "~/Views/Shared/_Layout.cshtml";
```

```
}


<h2>New Customer</h2>

@using (Html.BeginForm("Save", "Customer"))

{

    <div class="form-group">

        @Html.LabelFor(m=>m.Customer.Name)

        @Html.TextBoxFor(m => m.Customer.Name, new {@class="form-control" })

    </div>

    <div class="form-group">

        @Html.LabelFor(m => m.Customer.BirthDate)

        @Html.TextBoxFor(m => m.Customer.BirthDate,"{ 0:d MMM yyyy}", new { @class =
"form-control" })

    </div>

    <div class="checkbox">

        <label>

            @Html.CheckBoxFor(m=>m.Customer.IsSubscribedToNewsletter) Is Subscribed to
newsletter?

        </label>

    </div>

    <div class="form-group">

        @Html.LabelFor(m => m.Customer.MembershipTypeId)

        @Html.DropDownListFor(m => m.Customer.MembershipTypeId,

       new SelectList (Model.MembershipTypes,"Id","Name" ),

         "--Select--", new { @class = "form-control" })

    </div>

    <button type="submit" class="btn btn-primary">Save</button>

}
```

CustomerController.cs

```
[HttpPost]
        public ActionResult Save(Customer customer)
        {
            if(customer.Id==0)
            {
                _context.Customers.Add(customer);
            }
            else
            {
                var customerInDb = _context.Customers.Single(c => c.Id == customer.Id);
                //TryUpdateModel(customerInDb);
                //This has some drawback,it open security holes in appln
                //Only users with specific privileges should update the data
                //WIth this a malicious user can update the data
                //If we want o update only ceratin properties , then we can use below approach
                //TryUpdateModel(customerInDb,new string[] { "Name","Email"});
                //Here problem is the magic string
                customerInDb.Name = customer.Name;
                customerInDb.BirthDate = customer.BirthDate;
                customerInDb.MembershipTypeId = customer.MembershipTypeId;
                customerInDb.IsSubscribedToNewsletter =
customer.IsSubscribedToNewsletter;
                //Also we can use Automapper as below
                //Mapper.Map(customer, customerInDb);

            }
            _context.SaveChanges();
```

```
            return RedirectToAction("Index", "Customer");
        }
```

Interview:

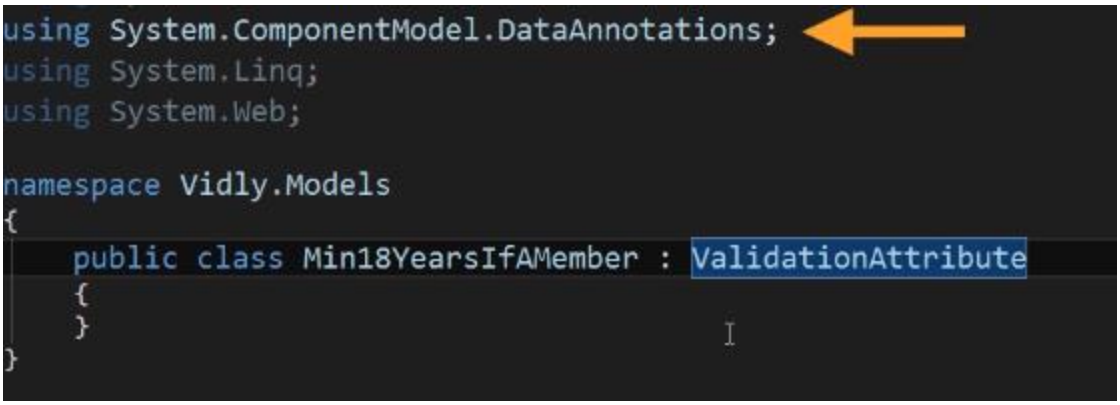
DataAnnotation :

Error message

```
[Required(ErrorMessage = "Please enter customer's name.")]
```

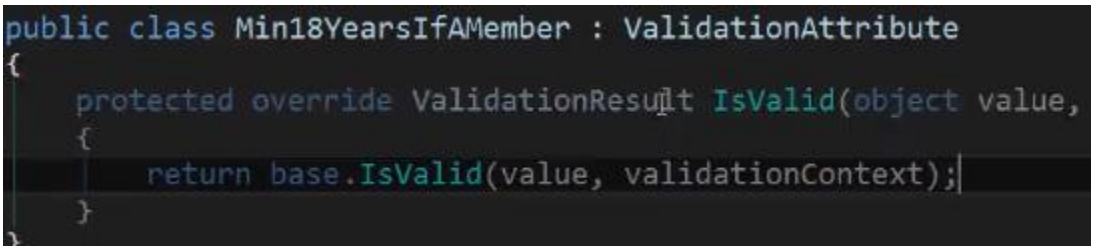
- [Required]
- [StringLength(255)]
- [Range(1, 10)]
- [Compare("OtherProperty")]
- [Phone]
- [EmailAddress]
- [Url]
- [RegularExpression("…")]

Custom Validation :

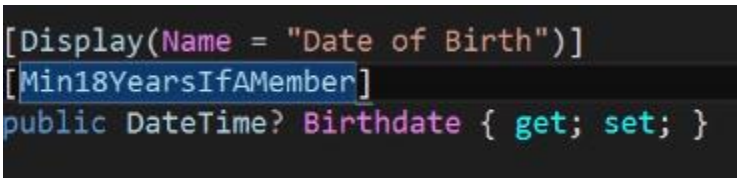


We need to inherit the class from ValidationAttribute

Namespace : System.ComponentModel.DataAnnotations



Then override IsValid



```
public class Min18YearsIfAMember : ValidationAttribute
{
    protected override ValidationResult IsValid(object value, Valida
    {
        var customer = (Customer)validationContext.ObjectInstance;

        if (customer.MembershipTypeId == 1)
            return ValidationResult.Success;

        if (customer.Birthdate == null)
            return new ValidationResult("Birthdate is required.");

        var age = DateTime.Today.Year - customer.Birthdate.Value.Yea

        return (age >= 18) ? ValidationResult.Success : new Validati
    }
}
```

var customer=(Customer)ValidationContext.ObjectInstance;

return ValidationResult.Success;

Validation Summary:

@Html.ValidationSummary();

Client side validation :

We can add Jquey validation bundle in the poage

DataAnnotations will be used for both client side and server side validation

```
@section scripts
{
    @Scripts.Render("~/bundles/jqueryval")
}
```

@sectio scripts

{

    @Scripts.Render("~/bundles/jqueryval")

}

AntiForgery Token :

[ValidateAntiForgeryToken]

@Html.AntiForgeryToken()

Avoid Cross Site REquest Forgery

It uses hidden filed

Hacker should not have access to this

Building WebAPI:

Insteadof returning html markup server can return raw data

Benefit: It requires only less server resources

Can increase the scalability of the application, and each client is responsible for generating thier own views

It requires only less bandwidhth, that improves the performance

Bit more respponsive

various types of clients can use this

Other websites can consume our webapi and can provide new functionality.

Many popular websites expose public data services which we can consume in our applications

we can use to modify the data base

WEbapi following the same architectural principles of mvc

Jquery plugin :Data table

Create folder WEbaPI-> Add controller ->Webapi2 controller- >

Configure webapi

Global.asax -> ApplicationStart()->

Add ->GlobalConfiguration.Configure(WebApiConfig.Register);

Namespace : System.Web.Http;

Api Controller class derives from: ApiController

/It will return a list of objects, by convention it will repond to

get/Api/Customers

public IEnumerable<Customer> GetCustomers()

{

```
        return _context.Customers.ToList();
}
Get/api/Customers/1
public Customer GetCustomer(int id)
{
        var customer= _context.Customers.SingleOrDefault(c=>c.Id==id);
        if(customer==null)
        {
                throw new HttpResponseException(HttpStatusCode.NotFound)
        }
        return customer

}

Insert Customer
[HttpPost]
publicCustomer CreateCustomer(Customer customer)
{
      if(!ModelState.IsValid)
      {
            throw new HttpResponseException(HttpSatatusCode.BadRequet);
            _context.Customers.Add(customer);
            _context.SaveCahnges();
           return customer;

      }
}

Update Customer
```

```
public void UpdateCustomer(int id, Customer customer)
{
    if (!ModelState.IsValid)
        throw new HttpResponseException(HttpStatusCode.BadRequest);

    var customerInDb = _context.Customers.SingleOrDefault(c => c.Id == id);

    if (customerInDb == null)
        throw new HttpResponseException(HttpStatusCode.NotFound);

    customerInDb.Name = customer.Name;
    customerInDb.Birthdate = customer.Birthdate;
    customerInDb.IsSubscribedToNewsletter = customer.IsSubscribedToNewslette
    customerInDb.MembershipTypeId = customer.MembershipTypeId;

    _context.SaveChanges();
}
```

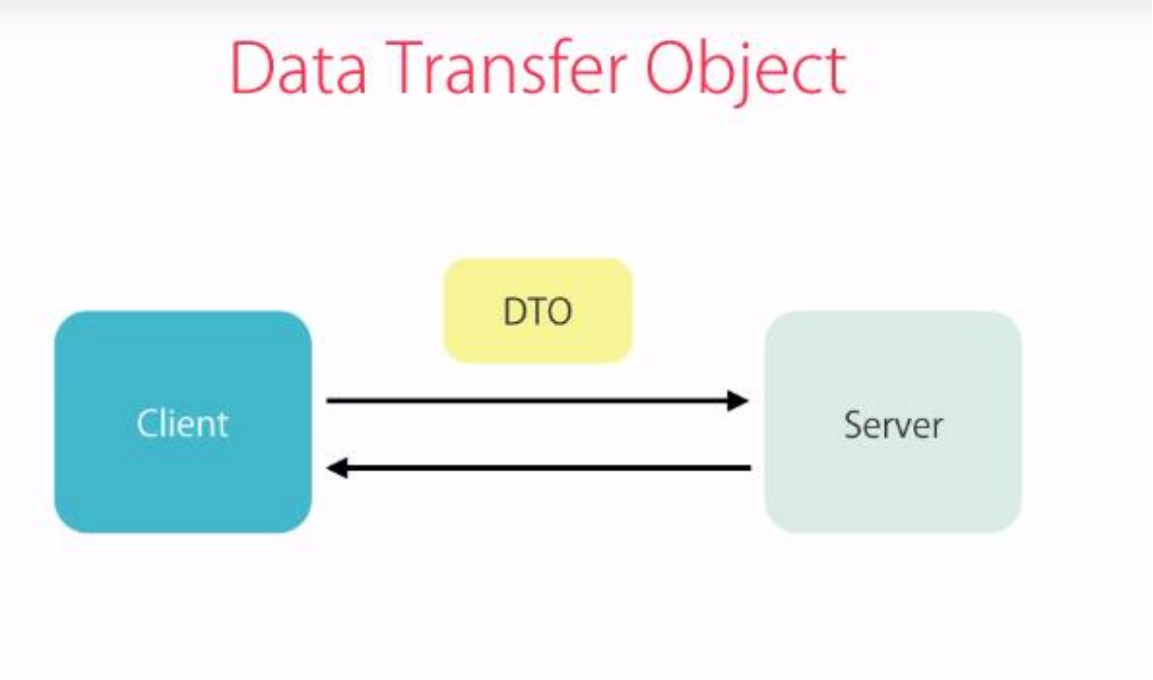
```
[HttpDelete]
public void DeleteCustomer(int id)
{
    var customerInDb = _context.Customers.SingleOrDefault(c => c.Id == id);

    if (customerInDb == null)
        throw new HttpResponseException(HttpStatusCode.NotFound);

    _context.Customers.Remove(customerInDb);
     context.SaveChanges();
```

DTO (Data Transfer Objects)

:

Webapi receives and sends domain model objects, which can change frequently

This change breaks the existing customer



DTO is a plane data structure which is used to transfer data from client to server and viceversa.

Usimg domain objects causing security holes in the application, hacker can easily add additional data into json and can easily map into domain object.

Creating DTO:

Create folder DTO -> create class CustomerDTO -> Paste all content from Customer class

AutoMapper :

PM> install-package automapper -version:4.1

Then in App_Start -> Add new class- > MappingProfile

MappingProfile.cs

using AutoMapper;

```
public class MappingProfile : Profile
{
    public MappingProfile()
    {
        Mapper.CreateMap<Customer, CustomerDto>();
        Mapper.CreateMap<CustomerDto, Customer>();
    }
}
```

automapper is using reflection to map

It finds its properties and maps them based on their name.

its a convention based mapping tool. since its using the proprty name to map the object.

Then in the Global.asax

in the ApplicationStart()

Mapper.Initialize(c=>c.AddProfile<MappingProfile>());

```
protected void Application_Start()
{
    Mapper.Initialize(c => c.AddProfile<MappingProfile>());
```

In trhe code: Everything need s to return and get from customerDTO

in GET

public IEnumerable<CustomerDto> GetCustomer()

{ _context.Customers.ToList().Select(Mapper.Map<Customer,CustomerDto>);

}

IHttpActionResult :

Here in webapi get will return 200 ok

but we need 201 ok

So in the get/api

we can change the action method as :

[HttpPost]

public HttpActionResult CreateCustomer(CustomerDto customerDto)

{ if(!ModelSatate.IsValid)

return BadRequest();

}

```
[HttpPost]
public IHttpActionResult CreateCustomer(CustomerDto customerDto)
{
    if (!ModelState.IsValid)
        return BadRequest();

    var customer = Mapper.Map<CustomerDto, Customer>(customerDto);
    _context.Customers.Add(customer);
    _context.SaveChanges();

    customerDto.Id = customer.Id;
    return Created(new Uri(Request.RequestUri + "/" + customer.Id), customer
}
```

return Created(new Uri(Request.RequestUri+ "/" + customer.Id),customerDto

;

calling webapi using Jquery :

```
<table class="table table-bordered table-hover">
    <thead>
        <tr>
            <th>Customer</th>
            <th>Membership Type</th>
            <th>Delete</th>
        </tr>
    </thead>
    <tbody>
        @foreach (var customer in Model)
        {
            <tr>
                <td>@Html.ActionLink(customer.Name, "Edit", "Customers", new {id
                <td>@customer.MembershipType.Name</td>
                <td>
                    <button class="btn-link">Delete</button>
                </td>
            </tr>
```

custoner table

element